PATENT
H26-083 US

ON-HARDWARE OPTIMIZATION OF STEPPER-MOTOR SYSTEM PERFORMANCE

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application is a continuation of provisional U.S. patent application no. 60/443,679, filed January 30, 2003, the entirety of which is hereby incorporated by reference.

FIELD AND BACKGROUND OF THE INVENTION

[0002] The present invention relates generally to the field of control systems and in particular to a new and useful controller for a stepper-motor.

[0003] Stepper-motors are ubiquitous in modern office equipment and other machinery, and yet little is published regarding their optimal use in open loop systems. Stepper motors are a good choice for open loop applications because the position error is not cumulative with rotation as long as synchroism is maintained.

[0004] Stepper-motors and control algorithms are disclosed for simulating the performance of particular operations. For example, Leenhouts, A.C., "Step Motor System Design Handbook, 2nd Ed.". Litchfield Engineering Co.: Kingman, AZ (1997) identifies algorithms for generating stepper motor step sequences and optimizing these sequences based on simulation of the stepper motor system. And, in Robrecht, M. and J. Luckel, "A Model-Based Approach to Generating an Optimized Sequence for Stepping-Motor Systems," 7th UK Mecahntronics Forum International Conference, Georgia Inst. of Tech., Atlanta, GA (2000) the step sequence is extracted from a closed loop stepper-driven system and applied in open loop mode.

[0005] Accurate control of a stepper-motor is limited by the ability of the control system to approximate the actual motor and subsequently provide the correct instructions for achieving the actual desired movement, with little or no error. That is, to move a motor to a specific position in a specific amount of time, and therefore at a certain velocity, is difficult due to variations between the motor and the model. The optimal solution may change depending on the desired optimized factor, such as minimizing residual vibration, producing a quick settling time or having an accurate position and velocity trajectory. While a solution can be found, finding the desired optimal performance solution for controlling a stepper-motor is difficult and time-consuming.

SUMMARY OF THE INVENTION

[0006] It is an object of the present invention to provide a method and apparatus for real-time optimization of a stepper-motor performance.

[0007] Accordingly, a control system for a stepper motor with real-time optimization is provided having computer-controlled interaction between optimization software and stepper-motor-driven system hardware. An interface or target computer mediates between a host computer sending control instructions and a hardware driver. The target computer interfaces between the optimizer and the hardware driver to provide the optimized step-time instructions to the motor. A step optimization model is loaded into the target computer for operating on the step-time sequence instructions to achieve an optimized step sequence based on an objective function.

[0008] The various features of novelty which characterize the invention are pointed out with particularity in the claims annexed to and forming a part of this disclosure. For a better understanding of the invention, its operating advantages and specific objects attained by its uses, reference is made to the accompanying drawings and descriptive matter in which a preferred embodiment of the invention is illustrated.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] In the drawings:

[0010] Fig. 1 is a schematic diagram of a stepper-motor test bed used with the invention;

[0011] Fig. 2 is a diagram illustrating the interaction of the components of the control system;

[0012] Fig. 3 is a diagram illustrating the programming hierarchy and information flow between the components of the control system;

[0013] Fig. 4A is a diagram of a step motor trajectory generator Simulink model;

[0014] Fig. 4B is a diagram of a further portion of the Simulink model of Fig. 4A;

[0015] Figs. 5A-5D are diagrams of components forming the step time optimization Simulink model used with the invention;

[0016] Fig. 6A is a graph plotting desired position against time for a control system optimized using the invention;

[0017] Fig. 6B is a graph plotting desired velocity against time used with the graph of Fig. 6A to defining a desired profile for test system;

[0018] Figs 7A & 7B are graphs showing results of an unoptimized test system trying to match the desired profile of

Figs. 6A and 6B; and

[0019] Figs. 8A & 8B are graphs showing optimized results for the test system seeking the desired profile of Figs. 6A and 6B.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0020] Referring now to the drawings, in which like reference numerals are used to refer to the same or similar elements, Fig. 1 shows a diagram of a motor test bed used with the invention. The motor test bed has a stepper motor 10 connected with an optical encoder 20 for sensing position of the motor and an inertial load 30. A driver 15 is connected to the stepper motor 10 for operating the motor 10 in accordance with the invention.

[0021] Experiments using this configuration were conducted on a motor test bed to demonstrate on-hardware optimization. The test bed arrangement of Fig. 1 was constructed using a bipolar stepper motor 10 to drive a rigidly coupled rotational inertial load 30 of 1.00 lbf-in^2, and the position is sensed with a Hewlett-Packard HEDM-5505-J06 incremental optical encoder 20. The motor 10 used in the experimental test bed was a Minebea 17PM-K101-03V. The motor 10 was driven in the bipolar configuration using an Intelligent Motion Systems' IM483 driver 15 to operate the motor 15 in half step mode at 24 volts.

A. Overview

[0022] The control system design process begins with a definition of the motion task. One such task might be to advance the load 18° and stop; another task might be to ramp up to a constant speed to drive the print head of an office printer across the paper while it prints, then ramp down to a stop. The motion task is translated into a position versus time or velocity versus time profile.

[0023] The profile is translated into a stepping command sequence to drive the stepper motor 10. Stepper motors driven by step-and-direction type drivers require two logic-level command signals. The direction input to the driver 15 controls the clockwise or counterclockwise rotation of the motor 10. The step input causes the motor 10 to advance one step for each low-to-high transition. Thus the stepping command signal takes the form of a pulse tram with as many pulses as steps needed for the desired motion.

[0024] An initial stepping sequence is generated by the Matlab Simulink model, such as that shown in Fig. 4A. The additional model components of Fig. 4B may be included as well. The acceleration profile required to obtain the desired position and velocity profiles is defined in this Simulink model. Acceleration, in steps per second-squared, is integrated and added to initial velocity to obtain the desired motor rotational rate in steps per second. Velocity is integrated to obtain the desired position in steps.

[0025] The model, referred to as a trajectory model, records the time whenever the desired position advances by an angle equal to one step. The step time sequence so formed is a column matrix of the times at which to step the motor; it represents the input to the stepper motor open loop system. The goal of the optimization system herein is to refine this step time sequence to get the optimum response from the motor 10.

B. Computer Software

[0026] The system is programmed using Matlab script files ("*.m" files) and Simulink models. The script files load parameters into the workspace, and control the programming flow during the optimization. Matlab uses the FMINCON command from the Optimization Toolbox to perform the optimization. This command is referred to herein as the optimizer. XPC Target is used as the real time interlace with the physical system. XPC Target requires Matlab Real Time Workshop, and a C-compiler.

[0027] A total of five script files are used. They are: top1.m, top2.m, nonlcon3.m, param1.m and runtest.m. The software files used to control the system are as follows.

[0028] **TOP1.M**

```
%top1.m
%Real time interactive optimization
clear all
global sd t y tg stepindex nsto steptimeopt scale
steptime st
```

```
st=.0001;
param1
motor17
%xeroxscan
%trap4
sim('traj20') %to generate trajectory files stepdir
and steptime
%steptime=[0;steptime;tout(end)];
%stepdir=[stepdir(1);stepdir;stepdir(end)] ;
sd=tout(end); %duration of simulation
%proceed = input('Build Model stepopt2, then enter Y
to proceed', 's');
'Build Model stepopt2 and run top2'
```

[0029] **TOP2.M**

```
%top2.m
%Real time interactive optimization
global steptimeopt nsto stepindex steptime scale st
tolc
%set scale to control initial stepsize in optimization
direction
%use scale=1e-5 for .0001s, use 1e-6 for .001; use
2*1e-6 for .0005
scale=5*1e-6;
%options=optimset('Display',    'final',    'TolX',
.001*scale, 'TolFun', .2);
tolc=.00005*scale;
options=optimset('Display',   'final',   'To   lX',
.0004*scale, 'TolCon', tolc);
A= [ ];
B= [ ];
```

```
Aeq= [ ] ;
Beq= [ ] ;
Lb=[.001; .002; .003; .004);
UB=[sd-.004;sd-.003;sd-.002;sd-.001);
Lbs=Lb*scale
UBs=UB*scale;
nonlconempty=[ ];
steptimeopt=[ ];
nsto=5; %number of steps to optimize at a time
%stepindex=2;
for   stepindex=16:nsto:length(steptime)   ;   %begin
optimizing at step 2. step 1 is always state=0, time=0
%for stepindex=2:nsto:6;
     stepindex
     %the  following  if  statement  governs  the  last
     iteration allowing the last steps
%to be optimized if they number less than nsto
if nsto+stepindex > length (steptime)
     nsto=length(steptime)-stepindex+1
     lastpass=1
end
steptimeopt(stepindex:stepindex+nsto-1, 1) = fmincon
('runtest',  steptime  (stepindex:stepindex + nsto-
1)*scale,  A,  B,  Aeq,  Beq,  Lbs,  UBs,  'non1con3',
options);
wk1write ('steprecord.wk1', steptimeopt/scale) ;
end

steptimeopt=steptimeopt/scale;
```

[0030] **NONLCON3.M**

```
%nonlcon file for nonlinear constraints
function [C, Ceq]= nonlcon3 (stimes)
global nsto stepindex steptimeopt scale
stime=stimes/scale;
% constrain step sequence to be monotonic with minimum
and maximum spacing
% compare the first step to the last optimized step
deltatmin=.001;
deltatmax=.007;
     if stepindex ==1
       TL=0; %Lower bound
     else
       TL=steptimeopt  (stepindex-1)/scale;  %Lower
     bound
     end
%C(1) is a lower limit on the first step being
optimized
C(1) = TL+deltatmin-stime(1);
%C(nsto+1) is an upper limit on the first step being
optimized
C(nsto+1)=stime(1)-TL-deltatmax;
if (nsto>1)
     for i=2; nsto
     %C(i) limits how close together any two steps
     may be
     C(i)=stime(i-1)-stime(i)+deltatmin;
     end
          for k=nsto+2:2*nsto
          %C(k) limits how far apart any two steps
          may be
          C(k)=stime(k-nsto)-stime(k-nsto-1)-
          deltatmax;
          end
end
Ceq=[ ]; %no quality constraints
```

[0031] **PARAM1.M**

```
%param1.m
%clear all
%global st sd t y  %tg
sd=inf; %simulation duration (stop time)
Ts=st;
%Gurley R135 encoder:
gurleylines=3600 % number of encoder counts per
revolution before quadrature
gurleyinterp=10 % encoder internal CPR multiplier

%HP HEDS-5645#I06 encoder:
lines=512
interp=1

%HP HEDS-5645#J06 encoder:
%lines=1024;;
%interp=1;

%Inkjet linear encoder:
%lines=150; % per inch
%interp=1;

%parameters for trapezoidal velocity profile
v0=300; % initial velocity, steps per second
```

```
%for traj48
%arate=5000; %acceleration rate, Hz/sec
%at=.05; %acceleration duration
%tt=.11; %total time
% for traj32
%arate=7000; %acceleration rate, Hz/sec
%at=.0286; %acceleration duration
%tt=.052; %total time
% for traj inkjet: 600 steps and reverse
%arate=8000;
%at=.0875;
%tt=.661;
%dwell=.1; %pause time at end of stroke
%settle=.1; %settling time at vmax before accumulating
objective function
% for traj20 20 steps
arate=7000;
at=.0286;
tt=.052;
```

[0032] **RUNTEST.M**

```
function [aof] =runtest (stimes)
%function [x, t, y] =runtest
global  tg  sd  t  y  scale  steptime  stepindex  nsto
steptimeopt st tolc
f=nonlcon3(stimes)-tolc/scale;
aofc=0;
if sum(not(f<=0)) %AB check constraints.
     %not (f<0) .*f
     aofc=50; %penalty if constraints are not met
     'constraint not met' ;
     %else%AB
end %AB
stime=stimes/scale%scale back to time in seconds
%build step sequence:
if sum(stime) > sum(steptime+.2) %GG
     aof=1000
else%GG
steptime2= [ ] ;
%create a column of equal delta t values, equal to the
difference between the last
% of this set of steps to be optimized and the
corresponding unoptimized step
%stime(end)
% stepindex+nsto-1
%steptime(stepindex+nsto-1)
%(steptime(stepindex+nsto-1)-stime(end))
temp=ones(size(steptime))* (steptime (stepindex +
nsto-1)- stime (end));
steptime2=(steptime-temp); %Shift remainder of
%unoptimized trajectory by delta t
% overwrite unoptimized with optimized and current:
steptime2(1:stepindex+nsto-1)=[steptimeopt/scale;sti
me];
     %if steptime2/stepindex)<steptime2(stepindex-1)
     %AAA check for nonmonotonic
     %aof=1000
     %else %AAA
          look=.2;
          settletime=.1;
          n=10; %number of hardware runs to average
%tg.stoptime=stime (end) +look;
%tg.stoptime=.25;
tg.p7=steptime2; %update step sequence on real time
target
clf
i=1;
j=0;
while i<=n %AA
     tg.start; %start simulation running on real time
     target
     t0=clock; %record start time
          while  etime(clock,  t0)  <  tg.stoptime  +
          settletime%sd  %BB  pause  while  simulation
          runs etime(clock, t0);
          end %BB
     if strcmp(tg.CPUoverload, 'none') %CC
%x = tg.StateLog;
y1 = tg.OutputLogi -
t = tg.TimeLog;
subplot (311)
plot(t, y1(:,3))
hold on
%axis([0 .3 0 1])
subplot (312)
plot(t, y1(:,2))
hold on
%axis([0 .3 -10 15])
subplot (313)
plot(t, y1(:,4))
```

```
%axis([0 .3 0 10])
hold on
     %if y1(end)>2.5 %DD
     %    i=9999
     %else %DD
          if i==1 %EE
          y   =  tg.OutputLog;   %Record   aggregate
          objective function from target
          else %EE;
          y = y + tg.OutputLog; %Record aggregate
          objective function from target
          end %EE
     %y(end,3)
     % aofi(i)=y(end) ;%
     %    end %DD
     i=i+1;
else %CC
     j=j+1
          if j==10  %FF
          'overload'
          i=200;
          y(end)=9999;
          end %FF
     end %CC
end %AA
y=y./n;
%aof=round(sum(aofi)/n*2)/2
%aof=sum(aofi)/n
% aof=round(y(end)*10)/10
%y(floor(stime(end)/st), 4) %aof at last step
```

```
% y(end) %aof at end of simulation
aof1=y(floor(steptime(end)/st) ,4)% up 'til last step
aof2=(y(end)-y(floor(steptime(end)/st), 4)) %consider
only ringing after %last step
aof=aof1+aof2*10+aofc; %weigh ringing *10
aof=round(aof*10)/10 %round to nearest 0.1
%t = tg.TimeLog; %Record time vector
%subplot(311)
%plot(t, y(:,1), 'r'),
%axis([0 .3 0 1])
% subplot (312)
%plot(t, y(:,2), 'r')
%axis([0 .3 -10 15])
%subplot(313)
%plot(t, y(:,4), 'r')
%axis([0 .3 0 10])
%end %AAA
end%GG
%end%AB
```

[0033] There are two Simulink models used in the embodiment described herein. The Simulink models are Traj20.mdl and Stepopt2.mdl. The models are illustrated in Figs. 4A and B and Figs. 5A-5D, respectively.

C. Structure of the Hardware-Software System

[0034] The optimization is performed on a two-computer system. Figs. 2 and 3 illustrate the system and operating components.

[0035] Referring to Fig. 2, the host computer 100 governs the optimization procedure and is the user interface for building and modifying models and other files. The target computer 150 with its data acquisition (DAQ) card serves as the interface between the optimizer and the physical system, including motor 10 and load 30. A control application, such as the Stepopt2.mdl model file, is downloaded from the host computer and runs in real time on the target computer 150. The functions of the real time application are to construct the command signals from the step time sequence, to pass the logic-level voltage command signals to the physical system, to record data from sensors, and to evaluate an objective function based on the system response.

[0036] Fig. 3 illustrates some of the connections between the hardware and target computer 150 in greater detail. As shown, step and direction instructions are received from target computer 150 by driver 15. Power supply 25 provides power to the hardware components through the driver 15, including the motor 10. Optical encoder 20 is mounted with load 30 to sense the position and trasmit that information back to the target computer 150 as real-time feedback. The feedback data is used by the optimization program on the target computer to refine and optimize the step-time sequence for the motor 10 in accordance with the objective function.

D. Optimization Procedure

[0037] As noted above, the optimization problem solved by the invention is to find the set of step time values that results in a position/velocity trajectory of the physical system that is closest to the desired trajectory. But, rather than finding the minimum of a mathematical objective function as in classical optimization, the "objective function" in this case is derived from the dynamic response of a physical system. The block diagram of the system shown in Fig. 3. best illustrates the optimization process.

[0038] In Fig. 3, three top-level commands are executed by the user: Top1.m, Stepopt2.mdl, and Top2.m. The Top1.m command loads the trajectory parameters, Param1.m, and the motor parameters Motor17.m. Then, the Top1.m command generates the initial step time sequence, as defined in the Simulink model Traj20.mdl. This step time sequence containing the desired number of steps is will serve as the starting guess at the solution to the optimization problem. The Simulink model Stepopt2.mdl is downloaded as a C program to the target computer. The Stepopt2.mdl program defines the objective function, and defines the input and output through the DAQ. The Top2.m script controls the program flow for the optimization.

[0039] The optimizer starts the application running on the target computer, which passes the command signal to the physical system. The target application records the system response from an optical encoder 20, calculates the value of the objective function, and reports this value to the optimizer. The optimizer checks the step time sequence against the constraints, checks the termination criteria, and uses a sequential quadratic programming (SQP) method to determine the next perturbation of the step time sequence. The optimizer controls the iterations, sending perturbed sequences to the target application, and interpreting the objective function.

E. The Objective Function

[0040] The objective function is a measure of how closely the physical system response matches the desired response. A low value of the objective function correlates with a good match. Thus we minimize the objective function to improve the dynamic performance of the system. The optimization variable is the step-time sequence. The time of each step is perturbed by the optimizer, and objective function is evaluated for each perturbation to measure the effect on the system response. The result is a step-time sequence that most nearly matches the desired angular position and angular velocity versus time.

[0041] The objective function used in the example herein is a weighted sum of position and velocity error squared, integrated over the duration of tile trajectory. It is defined in the Simulink model illustrated by Figs. 5A-5D, and calculated within the optimization application that runs on the target computer. The position and velocity errors are first squared, and then weighted in a 10,000-to-1 ratio with theta carrying the higher weight. The weighted squared errors are integrated individually and then summed to obtain an aggregate objective function. Squaring the error gives greater weight to the function at the points that deviate most from the desired position and velocity. The weight ratio is used to bring the two components to the same order of magnitude.

[0042] The objective function is derived from the response of the physical system and therefore is subject to natural variations, even if the same step time sequence is executed. This situation is unlike calculating an objective function from a mathematical equation that, of course, is repeatable.

[0043] To account for the natural variation, the same step time sequence is executed on the hardware repeatedly, 10 times in the current example, and the objective function value reported is the average of the 10 individually compute objective function values. The reported objective function value is rounded to a number of significant figures to further increase its repeatability.

Example

[0044] The optimization procedure was performed on the test bed described. The motion task in this example is to move twenty 0.90 half steps with minimal residual vibration. The desired motion profile is defined by desired position and desired velocity. The desired position and desired velocity are shown by Figs. 6A and 6B, respectively.

[0045] A corresponding step time sequence is generated from the desired trajectory. This is the initial unoptimized step time sequence created by the Traj20.mdl Simulink model, and is listed in the left hand column of the table below. The test bed response to the unoptimized command signal is shown in Figs. 7A and 7B. Stepper motors are known to exhibit low damping, and this case is no exception: the residual oscillation due to the static torque is plainly apparent in Figs. 7A and 7B.

[0046] The optimization procedure was applied to the system. The following table shows the initial, or unoptimized, and resulting optimized step time sequence values for the system:

Initial Step Time	**Optimized Step Time**
0.0020	0.0013
0.0048	0.0029
0.0080	0.0063
0.0108	0.0118
0.0132	0.0128
0.0156	0.0138
0.0180	0.0159
0.204	0.0186
0.0228	0.0205
0.0248	0.0225
0.0268	0.0268
0.0292	0.0278
0.0312	0.0288

0.0336	0.0298
0.0360	0.0312
0.0384	0.0358
0.0412	0.0378
0.0440	0.0419
0.0468	0.0456
0.0496	0.0529

[0047] The dramatic reduction in residual oscillation is apparent when comparing the unoptimized case of Figs. 7A & 7B to the optimized case of Fig. 8A & 8B.

[0048] The direct-on-hardware procedure yields a custom-optimized, machine-specific step state sequence. The goal in this example was to follow the trajectory in both position and velocity, and this is reflected in the objective function. A further goal was to reduce the residual vibration. The objective function addresses both goals because it is based on an integral over the nominal trajectory as well as the time to settle. Thus any residual vibration adds to the objective function value, and the optimization tends to decrease this residual vibration.

[0049] Objective functions can be tailored to the requirements of any motion task. For example, the goal of an inkjet printer system might be to maintain constant velocity while the print head is printing. In this case, the ramp up to speed, and the ramp down to stop need not be considered in the objective function.

[0050] Research on model-based optimization methods is also underway in addition to the direct optimization method described here. Model-based techniques use simulations rather than hardware measurements to provide feedback to the optimizer. Model-based methods are suitable to the development of systems because the designer can experiment with parameter ranges and gain an understanding of the relationships between, and effects of system parameters.

[0051] While a specific embodiment of the invention has been shown and described in detail to illustrate the application of the principles of the invention, it will be understood that the invention may be embodied otherwise without departing from such principles.